

DSC 140A

Probabilistic Modeling & Machine Learning

Lecture 8 | Part 1

High-Dimensional Feature Maps

Linear Prediction Rules

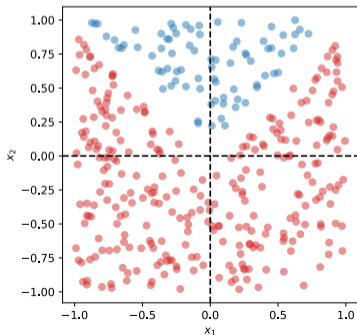
- ▶ We have seen how to fit linear functions:

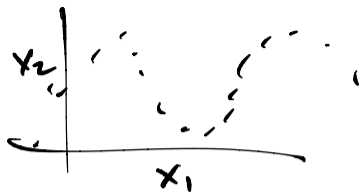
$$H(\vec{X}) = w_0 + w_1X_1 + \dots + w_dX_d$$

- ▶ Used for both **regression** and **classification**
- ▶ **Limitation:** regression function / decision boundary is a straight line / plane / hyperplane

Example

- ▶ The data below is not **linearly separable**
- ▶ No prediction function of the form $H(x_1, x_2) = w_0 + w_1x_1 + x_2x_2$ will work well

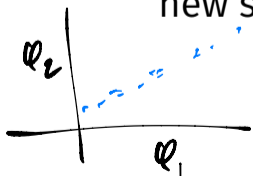




$$H(\vec{x}) = w_0 + w_1 \varphi_1(\vec{x}) + w_2 \varphi_2(\vec{x}) + \dots$$

However...

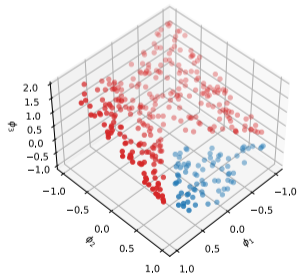
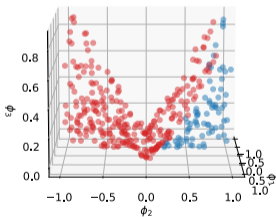
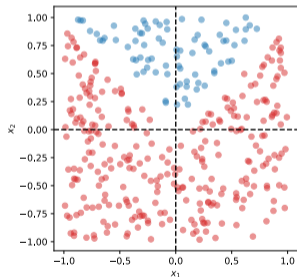
- ▶ We have seen a way around this limitation: **basis functions**.
- ▶ **Idea:** design a function $\vec{\phi}(\vec{x})$ that maps data to a new space in which it is **linearly separable**.



$$w_2 + w_1 \phi_1 + w_2 \phi_2 + w_3 \phi_3$$

Example

- ▶ Consider the mapping $\vec{\phi}(x_1, x_2) = (x_1, x_2, |x_1 x_2|)^T$



$$Q_1 = 1$$

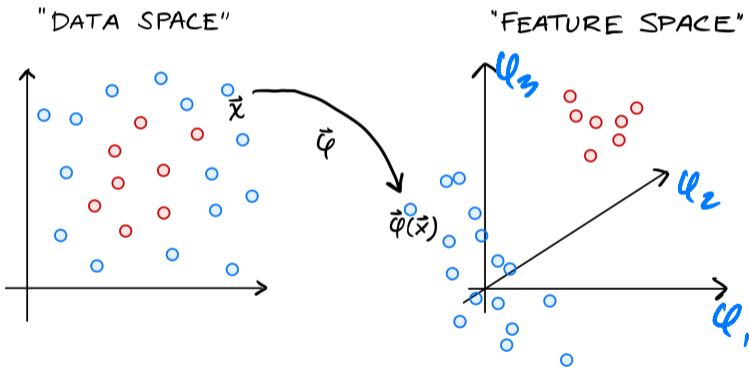
Procedure

1. Define feature map $\vec{\phi}(\vec{x}) : \mathbb{R}^d \rightarrow \mathbb{R}^k$
 - ▶ $\vec{\phi}(\vec{x}) = (\phi_1(\vec{x}), \dots, \phi_k(\vec{x}))^T$
 - ▶ Number of basis functions k can be $>$ or \leq than d
2. Map each training point to k -dimensional **feature space**: $\vec{x}^{(i)} \mapsto \vec{\phi}(\vec{x}^{(i)})$
3. Learn a linear predictor in feature space:

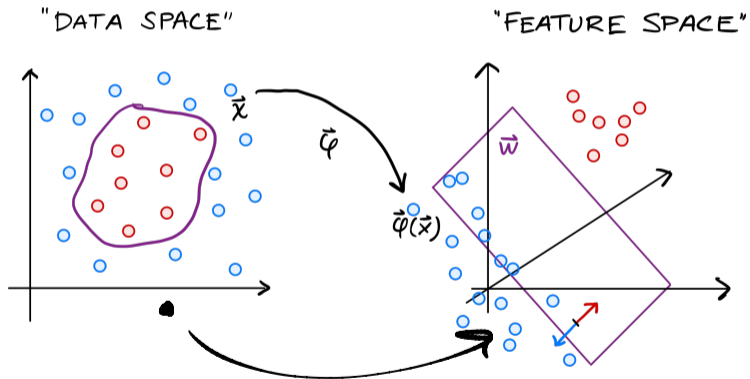
$$H(\vec{x}) = \cancel{w_0} + w_1 \phi_1(\vec{x}) + \dots + \overset{w_k}{\phi_k(\vec{x})}$$

Procedure

$$H(\vec{x}) = w_0 + w_1 \varphi_1(\vec{x}) + w_2 \varphi_2(\vec{x}) + w_3 \varphi_3(\vec{x})$$



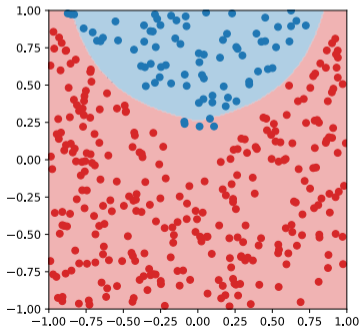
Procedure



Example

- ▶ Use mapping $\vec{\phi}(\vec{x}) = (x_1, x_2, |x_1 x_2|)^T$
- ▶ Decision boundary in “data space” no longer a straight line.

$$H(\vec{\phi}) = w_0 + w_1 x_1 + w_2 x_2 + w_3 |x_1 x_2|$$



$$\vec{\varphi}(\vec{x}) = \vec{\varphi}(2, -3) = (2, -3, 6)^T$$

$$H(\vec{x}) = \vec{w} \cdot$$

Exercise

Suppose $\vec{w} = (3, -1, 2)^T$ defines a linear predictor in feature space and $\vec{\phi} = (x_1, x_2, |x_1 x_2|)^T$ is the mapping from "data space" to "feature space".

Let $\vec{x} = (2, -3)^T$ be a new point that needs to be classified. What is the predicted label?

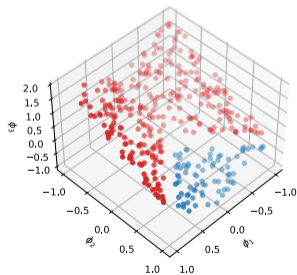
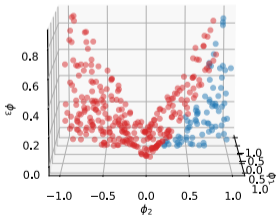
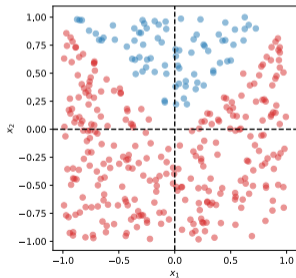
$$\begin{matrix} \vec{w} \\ \left(\begin{array}{c} 3 \\ -1 \\ 2 \end{array} \right) \end{matrix} \cdot \begin{matrix} \varphi(\vec{x}) \\ \left(\begin{array}{c} 2 \\ -3 \\ 6 \end{array} \right) \end{matrix} = \begin{matrix} +6 \\ +3 \\ 12 \end{matrix} = 21 \quad H(\vec{x}) = 21$$

Feature Maps

- ▶ How do we choose $\vec{\phi}$?
- ▶ **Hope:** data is linearly separable in feature space
- ▶ Appears difficult to engineer $\vec{\phi}$ to satisfy this.
 - ▶ Need to design $\vec{\phi}$ for each new data set?
- ▶ **Goal:** design a general feature map that is likely to make any data set linearly separable

High-Dimensional Feature Maps

- **Observe:** in our example, ϕ mapped to space of larger dimension



High-Dimensional Feature Maps

- ▶ **Intuition:** each additional feature makes the data easier to classify.
- ▶ **Intuition:** a high-dimensional feature map is likely to make the data linearly separable.
- ▶ **Idea:** design *very* high-dimensional generic feature maps.

(x_1, x_2)

Example: Monomials

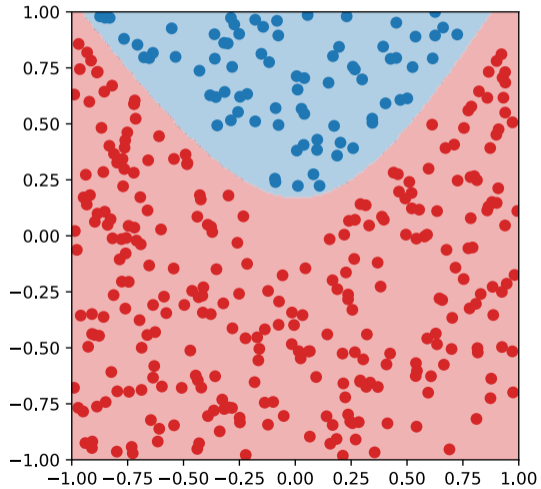
- ▶ Define a feature map $\vec{\phi} : \mathbb{R}^2 \rightarrow \mathbb{R}^6$ as follows:

$$\vec{\phi}(\vec{x}) = (1, x_1, x_2, x_1x_2, x_1^2, x_2^2)^T$$

- ▶ We fit a prediction function of the form:

$$H(\vec{x}) = w_0 + w_1x_1 + w_2x_2 + w_3x_1x_2 + w_4x_1^2 + w_5x_2^2$$

Example: Monomials



$(x_1, x_2, x_3) \mapsto (1, x_1, x_2, x_3, x_1x_3, x_2x_3, x_1x_2, x_1^2, x_2^2, x_3^2)$
Example: Monomials

- ▶ In general, define a feature map $\vec{\phi}$ to contain all **monomials** of the form:

$$1, \quad x_i, \quad x_i x_j, \quad x_i^2$$

- ▶ If $\vec{x} \in \mathbb{R}^d$, then $\vec{\phi}(\vec{x}) \in \mathbb{R}^{1+2d+\binom{d}{2}}$.
- ▶ **Example:** if $\vec{x} \in \mathbb{R}^{50}$, then $\vec{\phi}(\vec{x}) \in \mathbb{R}^{1,326}$.

Example: Monomials

- ▶ Why stop there? Design $\vec{\phi}$ to contain all terms of form:

$$1, \quad x_i, \quad x_i x_j, \quad x_i^2, \quad x_i x_j x_k, \quad x_i^3$$

- ▶ If $\vec{x} \in \mathbb{R}^d$, then $\vec{\phi}(\vec{x}) \in \mathbb{R}^{1+3d+\binom{d}{2}+\binom{d}{3}}$.
- ▶ **Example:** if $\vec{x} \in \mathbb{R}^{50}$, then $\vec{\phi}(\vec{x}) \in \mathbb{R}^{20,976}$!
- ▶ And so on...

Problem

- ▶ Mapping to very high dimensions is likely to make the data linearly separable.
- ▶ But fitting a linear prediction rule in high dimensions is **costly**.

DSC 140A

Probabilistic Modeling & Machine Learning

Lecture 8 | Part 2

The Kernel Trick

Recap

- ▶ We can learn non-linear patterns by:
 1. Defining a high-dimensional feature map,
 $\vec{\phi} : \mathbb{R}^d \rightarrow \mathbb{R}^k$
 2. Mapping each training point to k -dimensional **feature space**: $\vec{x}^{(i)} \mapsto \vec{\phi}(\vec{x}^{(i)})$
 3. Training a linear predictor in feature space.

Problem

- ▶ Learning in a very high-dimensional space can be costly, or even infeasible.

The Trick

- ▶ We can train many linear predictors *as if* we have mapped data to feature space, **without actually doing so.**

Idea

- ▶ In many algorithms, when $\vec{\phi}(\vec{x})$ appears, it always appears as part of a dot product:

$$\vec{\phi}(\vec{x}) \cdot \vec{\phi}(\vec{x}')$$

- ▶ To compute, we *could* map and do dot product in feature space.
- ▶ But this is **costly!**

Kernels

- ▶ But some $\vec{\phi}$ are special; for them, there is a function κ satisfying:

$$\kappa(\vec{x}, \vec{x}') = \vec{\phi}(\vec{x}) \cdot \vec{\phi}(\vec{x}')$$

- ▶ Crucially, computing κ does **not require mapping to feature space!**
- ▶ κ is called a **kernel** function.

The Kernel Trick

- ▶ In many algorithms, when $\vec{\phi}(\vec{x})$ appears, it always appears as part of a dot product of the form:

$$\vec{\phi}(\vec{x}) \cdot \vec{\phi}(\vec{x}')$$

- ▶ By replacing all instances of $\vec{\phi}(\vec{x}) \cdot \vec{\phi}(\vec{x}')$ with $\kappa(\vec{x}, \vec{x}')$, we **kernelize** the algorithm; avoid mapping to feature space.
- ▶ This is called the **kernel trick**.

Example: Polynomial Kernel

- ▶ Define the feature map:

$$\vec{\phi}(\vec{x}) = (1, x_1^2, x_2^2, x_3^2, \sqrt{2} x_1, \sqrt{2} x_2, \sqrt{2} x_3, \sqrt{2} x_1 x_2, \sqrt{2} x_1 x_3, \sqrt{2} x_2 x_3)^T$$

- ▶ $\kappa(\vec{x}, \vec{x}') = (1 + \vec{x} \cdot \vec{x}')^2$ is a kernel for this $\vec{\phi}$.
 - ▶ That is, $\kappa(\vec{x}, \vec{x}') = \vec{\phi}(\vec{x}) \cdot \vec{\phi}(\vec{x}')$
- ▶ Called the **polynomial kernel**¹

¹In general, $\kappa(\vec{x}, \vec{x}') = (1 + \vec{x} \cdot \vec{x}')^k$ is kernel for k -order monomial mappings

Kernelized Algorithms

- ▶ Only certain mappings have efficiently-computed kernels.
- ▶ Only certain learning algorithms can be **kernelized**.
- ▶ **All** of the linear algorithms we've learned can.
 - ▶ Least squares, perceptron, SVMs, etc.

Kernel Ridge Regression

- ▶ Let's kernelize **ridge regression**.
- ▶ First: verify that all instances of $\vec{\phi}(\vec{x})$ appear as part of a dot product: $\vec{\phi}(\vec{x}) \cdot \vec{\phi}(\vec{x}')$

Kernel Ridge Regression

- ▶ Suppose $\vec{\phi}(\vec{x})$ is a feature map with kernel k .
- ▶ To train a ridge regressor in feature space, we'd solve

$$\arg \min_{\vec{w}} \frac{1}{n} \sum_{i=1}^n \left(\vec{\phi}(\vec{x}^{(i)}) \cdot \vec{w} - y_i \right)^2 + \lambda \|\vec{w}\|^2$$

- ▶ In matrix-vector form, where Φ is the design matrix:

$$\arg \min_{\vec{w}} \frac{1}{n} \|\Phi \vec{w} - \vec{y}\|^2 + \lambda \vec{w}^T \vec{w}$$

Fact

- ▶ The solution w^* is a linear combination of $\vec{\phi}(\vec{x}^{(i)})$:

$$\vec{w}^* = \sum_{i=1}^n \alpha_i \vec{\phi}(\vec{x}^{(i)})$$

- ▶ Why? The gradient of the regularized risk is:

$$\frac{2}{n} \sum_{i=1}^n \left(\vec{\phi}(\vec{x}^{(i)}) \cdot \vec{w} - y_i \right) \vec{\phi}(\vec{x}^{(i)}) + 2\lambda \vec{w}$$

- ▶ Setting to zero, solving for \vec{w} gives:

$$\vec{w}^* = \sum_{i=1}^n \underbrace{\left(-\frac{1}{n\lambda} \vec{\phi}(\vec{x}^{(i)}) \cdot \vec{w}^* - y_i \right)}_{\alpha_i} \vec{\phi}(\vec{x}^{(i)})$$

$$\Phi = \begin{pmatrix} \phi(\vec{x}^{(1)}) \longrightarrow \\ \phi(\vec{x}^{(2)}) \longrightarrow \\ \vdots \end{pmatrix}$$

Fact

- ▶ The solution w^* is a linear combination of $\vec{\phi}(\vec{x}^{(i)})$:

$$\vec{w}^* = \sum_{i=1}^n \alpha_i \vec{\phi}(\vec{x}^{(i)})$$

- ▶ In matrix-vector form, where $\vec{\alpha} = (\alpha_1, \dots, \alpha_n)^T$:

$$\vec{w}^* = \Phi^T \vec{\alpha}$$

Dual Problem

$$\vec{w} = \Phi^T \vec{\alpha}$$

- ▶ Using the fact that $\vec{w}^* = \sum_{i=1}^n \alpha_i \vec{\phi}(\vec{x}^{(i)}) = \Phi^T \vec{\alpha}$ for some $\vec{\alpha}$, the problem:

$$\arg \min_{\vec{w}} \frac{1}{n} \|\Phi \vec{w} - \vec{y}\|^2 + \lambda \vec{w}^T \vec{w}$$

is equivalent to the **dual** problem:

$$\arg \min_{\vec{\alpha}} \frac{1}{n} \|\Phi \Phi^T \vec{\alpha} - \vec{y}\|^2 + \lambda \vec{\alpha}^T \Phi \Phi^T \vec{\alpha}$$

Kernelizing

- ▶ Where does $\vec{\phi}(\vec{x})$ appear in this problem?

$$\arg \min_{\vec{\alpha}} \frac{1}{n} \|\Phi \vec{\alpha} - \vec{y}\|^2 + \lambda \vec{\alpha}^T \Phi^T \vec{\alpha}$$

(Handwritten annotations: 'K' with arrows pointing to the Φ terms in the equation)

$$K(\vec{x}, \vec{x}') = \phi(\vec{x}) \cdot \phi(\vec{x}')$$

- ▶ Inside Φ :

$$\Phi = \begin{pmatrix} \vec{\phi}(\vec{x}^{(1)}) & \longrightarrow & \\ \vec{\phi}(\vec{x}^{(2)}) & \longrightarrow & \\ \vdots & & \\ \vec{\phi}(\vec{x}^{(n)}) & \longrightarrow & \end{pmatrix}$$

$(AB)_{ij}$

Exercise

Argue that the (i, j) entry of $\Phi\Phi^T$ is equal to $\kappa(\vec{x}^{(i)}, \vec{x}^{(j)})$.

$$\Phi = \begin{pmatrix} \vec{\phi}(\vec{x}^{(1)}) & \longrightarrow & \\ \vec{\phi}(\vec{x}^{(2)}) & \longrightarrow & \\ \vdots & & \\ \vec{\phi}(\vec{x}^{(n)}) & \longrightarrow & \end{pmatrix} \begin{pmatrix} \phi(\vec{x}^{(1)}) & \vec{\psi}(\vec{x}^{(2)}) & \dots \\ \downarrow & \downarrow & \\ \downarrow & \downarrow & \end{pmatrix}$$

Kernelizing

- ▶ The (i, j) entry of $\Phi\Phi^T$ is $\vec{\phi}(\vec{x}^{(i)}) \cdot \vec{\phi}(\vec{x}^{(j)}) = \kappa(\vec{x}^{(i)}, \vec{x}^{(j)})$

$$\Phi\Phi^T = \underbrace{\begin{pmatrix} \kappa(\vec{x}^{(1)}, \vec{x}^{(1)}) & \kappa(\vec{x}^{(1)}, \vec{x}^{(2)}) & \dots & \kappa(\vec{x}^{(1)}, \vec{x}^{(n)}) \\ \kappa(\vec{x}^{(2)}, \vec{x}^{(1)}) & \kappa(\vec{x}^{(2)}, \vec{x}^{(2)}) & \dots & \kappa(\vec{x}^{(2)}, \vec{x}^{(n)}) \\ \vdots & \vdots & \ddots & \vdots \\ \kappa(\vec{x}^{(n)}, \vec{x}^{(1)}) & \kappa(\vec{x}^{(n)}, \vec{x}^{(2)}) & \dots & \kappa(\vec{x}^{(n)}, \vec{x}^{(n)}) \end{pmatrix}}_K$$

- ▶ K is called the **Kernel matrix** (or **Gram matrix**).

Kernel Ridge Regression

- ▶ The dual problem becomes:

$$\arg \min_{\vec{\alpha}} \frac{1}{n} \|K\vec{\alpha} - \vec{y}\|^2 + \lambda \vec{\alpha}^T K \vec{\alpha}$$


- ▶ Exact solution:

$$\vec{\alpha}^* = (K + n\lambda I)^{-1} \vec{y}$$

- ▶ This is **kernel ridge regression**.

Kernelization

- ▶ **Observe:** we train linear predictor in feature space without actually mapping to feature space:

$$\vec{\alpha}^* = (K + \lambda I)^{-1} \vec{y}$$


A handwritten red arrow points from the λ term in the matrix $(K + \lambda I)$ to the expression $n\lambda I$ written below it.

Making Predictions

- ▶ To predict on a new point \vec{x} , normally:
 $H(\vec{x}) = \vec{w}^* \cdot \vec{\phi}(\vec{x})$.
- ▶ How to do this without actually mapping?

- ▶ Recall: $w^* = \sum_{i=1}^n \alpha_i^* \vec{\phi}(\vec{x}^{(i)})$

- ▶ So:

$$H(\vec{x}) = \sum_{i=1}^n \alpha_i^* \vec{\phi}(\vec{x}^{(i)}) \cdot \vec{\phi}(\vec{x}) = \sum_{i=1}^n \alpha_i^* \kappa(\vec{x}^{(i)}, \vec{x})$$

Making Predictions

- ▶ To make a prediction on a new point:

$$H(\vec{x}) = \sum_{i=1}^n \alpha_i^* \kappa(\vec{x}^{(i)}, \vec{x})$$

- ▶ No need to map to feature space.
- ▶ **Interpretation:** A weighted sum of kernel evaluations.

Procedure: Kernel Ridge Regression

1. Pick a kernel function, κ .

2. Solve linear system: $\vec{\alpha}^* = (K + \lambda I)^{-1} \vec{y}$

3. To make new prediction, $H(\vec{x}) = \sum_{i=1}^n \alpha_i^* \kappa(\vec{x}^{(i)}, \vec{x})$

Kernel Soft-SVM

► Soft-SVM can also be **kernelized**.

1. Pick a kernel function, κ .
2. Solve dual problem (e.g., with SGD):

$$\arg \min_{\vec{\alpha}} \left(\lambda \vec{\alpha}^T K \vec{\alpha} + \frac{1}{n} \sum_{i=1}^n \max\{0, 1 - y_i (K \vec{\alpha})_i\} \right)$$

3. To make new prediction, $H(\vec{x}) = \sum_{i \in S} \alpha_i^* \kappa(\vec{x}^{(i)}, \vec{x})$
 - Where S is the set of indices of support vectors.

Kernelization **Downsides**

- ▶ Often, training involves the $n \times n$ kernel matrix.
 - ▶ Can be very large!
- ▶ There are ways to mitigate this:
 - ▶ Small-batch stochastic gradient descent.
 - ▶ Nyström method.

DSC 140A

Probabilistic Modeling & Machine Learning

Lecture 8 | Part 3

Kernel Functions

Valid Kernels

- ▶ The first step in kernel learning is to pick a **kernel function**, κ .
- ▶ To be a valid kernel, must compute the dot product w.r.t., some mapping, $\vec{\phi}(\vec{x})$.
- ▶ That is, it must be that

$$\kappa(\vec{x}, \vec{x}') = \vec{\phi}(\vec{x}) \cdot \vec{\phi}(\vec{x}')$$

for some $\vec{\phi}$.

Constructing Kernels: Approach #1

- ▶ How do we come up with valid kernel functions?
- ▶ Approach #1:
 1. Start by picking $\vec{\phi}$
 2. Find a function κ that efficiently computes $\vec{\phi}(\vec{x}) \cdot \vec{\phi}(\vec{x}')$, if one exists.

Constructing Kernels: Approach #2

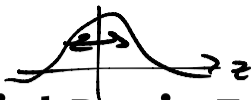
- ▶ New kernels can be constructed from other kernels.
- ▶ Suppose $\kappa_1, \kappa_2, \kappa_3$ are kernels and f is any function. Then the below are kernels:
 - ▶ $\kappa(\vec{x}, \vec{x}') = \kappa_1(\vec{x}, \vec{x}') + \kappa_2(\vec{x}, \vec{x}')$
 - ▶ $\kappa(\vec{x}, \vec{x}') = \kappa_1(\vec{x}, \vec{x}') \times \kappa_2(\vec{x}, \vec{x}')$
 - ▶ $\kappa(\vec{x}, \vec{x}') = \kappa_3(\vec{\phi}(\vec{x}), \vec{\phi}(\vec{x}'))$
 - ▶ $\kappa(\vec{x}, \vec{x}') = f(\vec{x})\kappa_1(\vec{x}, \vec{x}')f(\vec{x}')$

Verifying Kernels

Theorem

A symmetric function κ is a valid kernel if and only if the kernel matrix, K , is positive semi-definite for any choice of data, $\vec{x}^{(1)}, \dots, \vec{x}^{(n)}$.

$$e^{-z^2/\sigma^2}$$



Radial Basis Function Kernel

- ▶ Often, though, we don't design our own kernel.
- ▶ A very popular choice: the **radial basis function (RBF) kernel** (or **Gaussian kernel**):

$$\kappa(\vec{x}, \vec{x}') = e^{\frac{-\|\vec{x}-\vec{x}'\|^2}{2\sigma^2}} = e^{-\gamma\|\vec{x}-\vec{x}'\|^2} \quad \text{where } \gamma = 1/(2\sigma^2)$$

RBF Kernel Interpretation

$$\kappa(\vec{x}, \vec{x}') = e^{\frac{-\|\vec{x}-\vec{x}'\|^2}{2\sigma^2}} = e^{-\gamma\|\vec{x}-\vec{x}'\|^2}$$

- ▶ **Interpretation:** RBF kernel measures similarity of \vec{x} and \vec{x}'
 - ▶ Very similar: $\kappa(\vec{x}, \vec{x}') \approx 1$.
 - ▶ Very different: $\kappa(\vec{x}, \vec{x}') \approx 0$.
- ▶ Parameter σ (or γ) controls the scale
 - ▶ The larger σ (smaller γ), the wider the Gaussian

RBF Kernel Interpretation

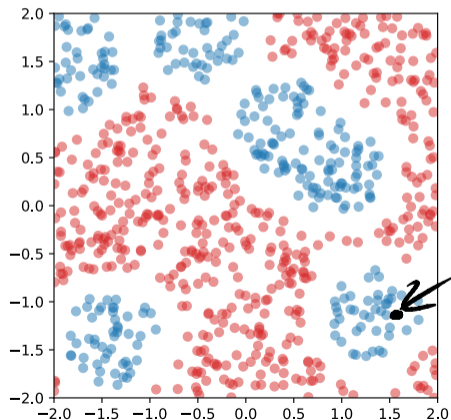
- ▶ Recall that in kernel ridge regression / SVM, the prediction is:

$$H(\vec{x}) = \sum_{i=1}^n \alpha_i K(\vec{x}^{(i)}, \vec{x})$$

- ▶ **Observations:**
 - ▶ One parameter α_i learned for **each** training point $\vec{x}^{(i)}$
 - ▶ $K(\vec{x}^{(i)}, \vec{x})$ will be ≈ 0 for any $\vec{x}^{(i)}$ far from \vec{x}
 - ▶ $H(\vec{x})$ is largely determined by the training points closest to \vec{x}

RBF Kernel Interpretation

- ▶ RBF function placed at each training point.
- ▶ $H(\vec{x})$ is largely determined by training points closest to \vec{x}



RBF Kernel Interpretation

- ▶ An RBF Kernel predictor can be seen as a generalization of the k -nearest neighbor rule

RBF Kernel Map

- ▶ What ϕ is the RBF kernel a kernel for?
- ▶ The mapping $\vec{\phi}(\vec{x})$ with entries of the form:

$$e^{-\|\vec{x}\|^2/2}x_i, \quad \frac{1}{\sqrt{2!}}e^{-\|\vec{x}\|^2/2}x_ix_j, \quad \frac{1}{\sqrt{3!}}e^{-\|\vec{x}\|^2/2}x_ix_jx_k, \quad \dots$$

- ▶ This is a mapping to an **infinite dimensional** space!

Other Kernels

- ▶ There are other interesting kernels useful for specific domains.
- ▶ **Example:** string kernels for text classification.
 - ▶ Dot product in space generated by all substrings.

DSC 140A

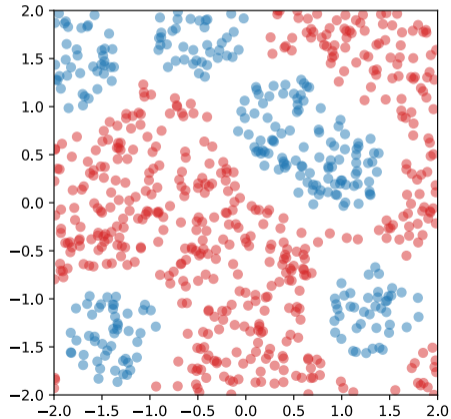
Probabilistic Modeling & Machine Learning

Lecture 8 | Part 4

Demo: Kernel SVM

Demo

- ▶ Train an RBF kernel SVM on the data below.



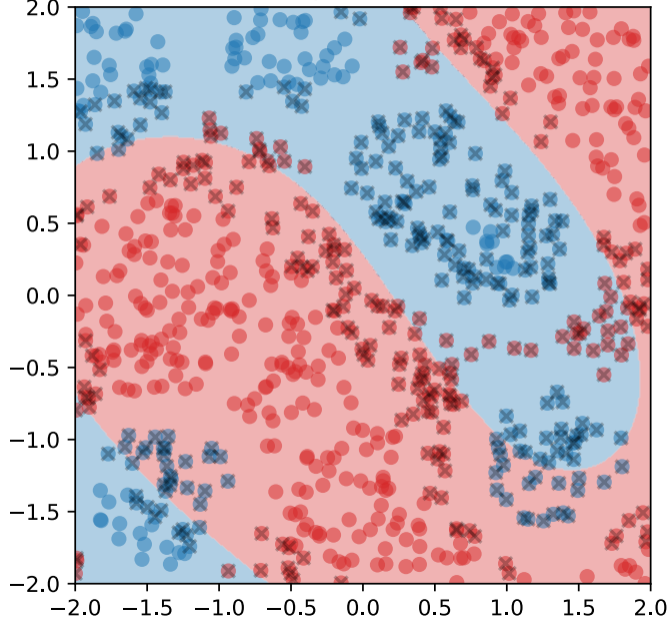
$$\frac{1}{2\sigma^2} = \gamma$$

Aside: Hyperparameter Selection

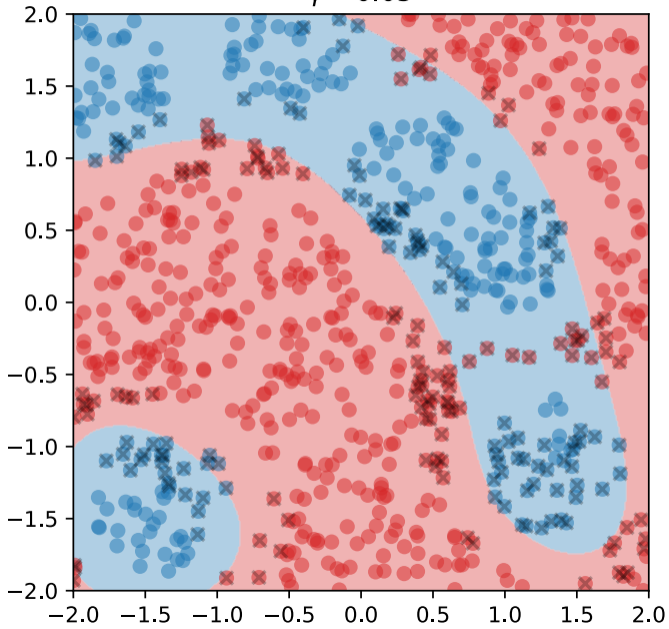
- ▶ Two hyperparameters to specify:
 - ▶ Slack: C
 - ▶ Kernel width: γ

- ▶ Choose with grid search cross-validation

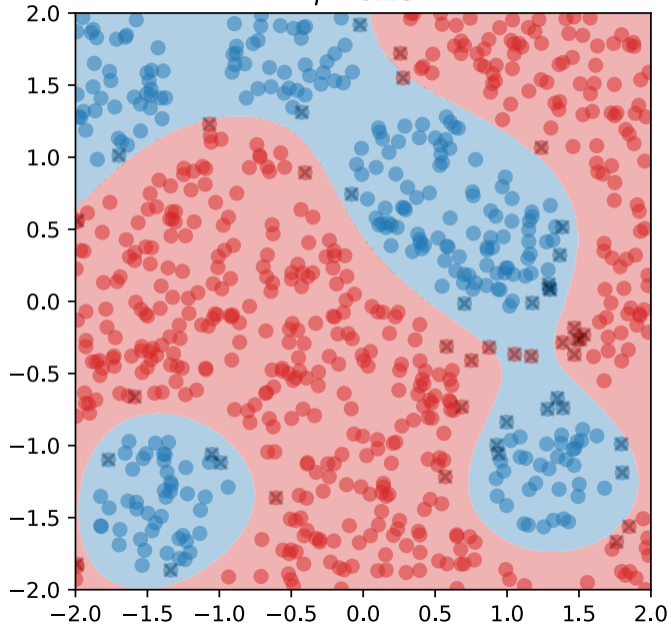
$$H(\vec{x}) = \sum_i \alpha_i \text{Gaussian}(\vec{x}^{(i)}, \vec{x}) \quad \gamma = 0.02$$



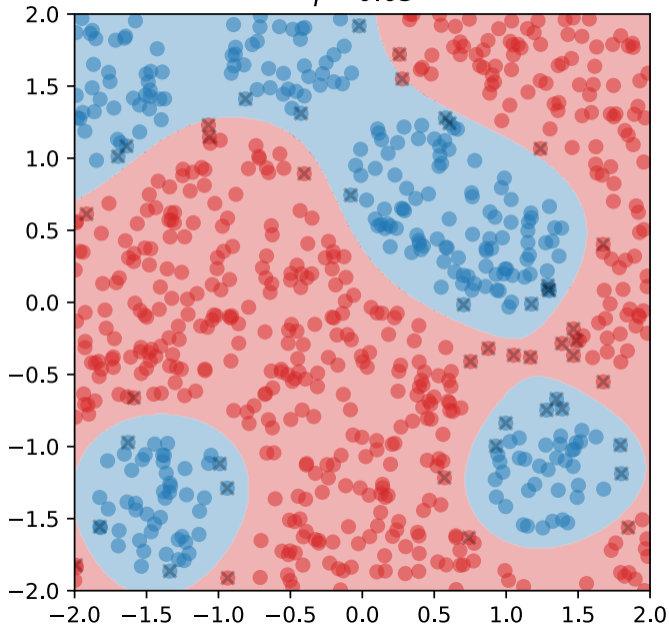
$\gamma = 0.05$



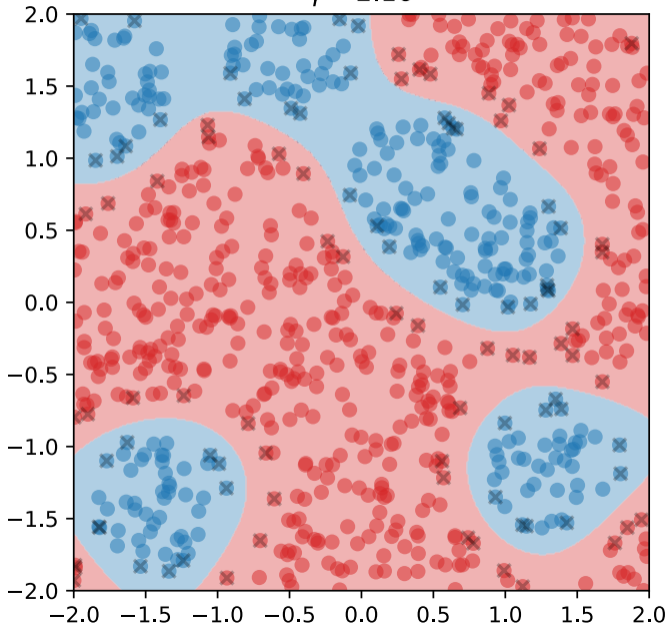
$\gamma = 0.18$



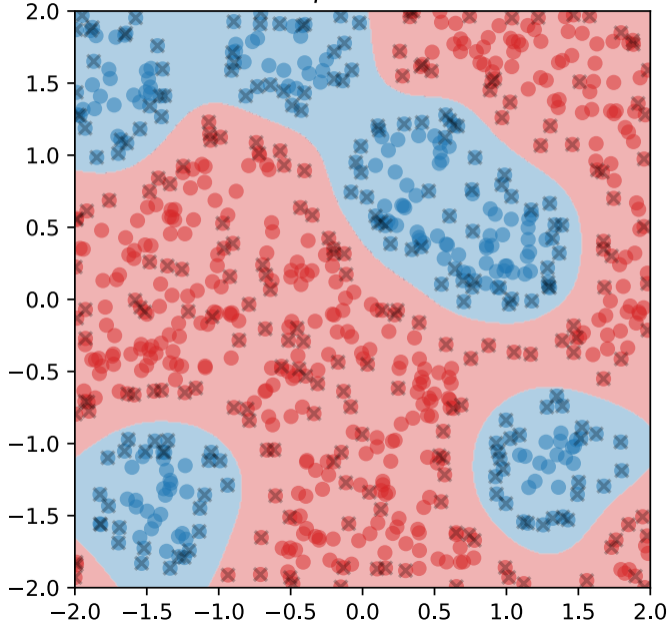
$\gamma = 0.63$



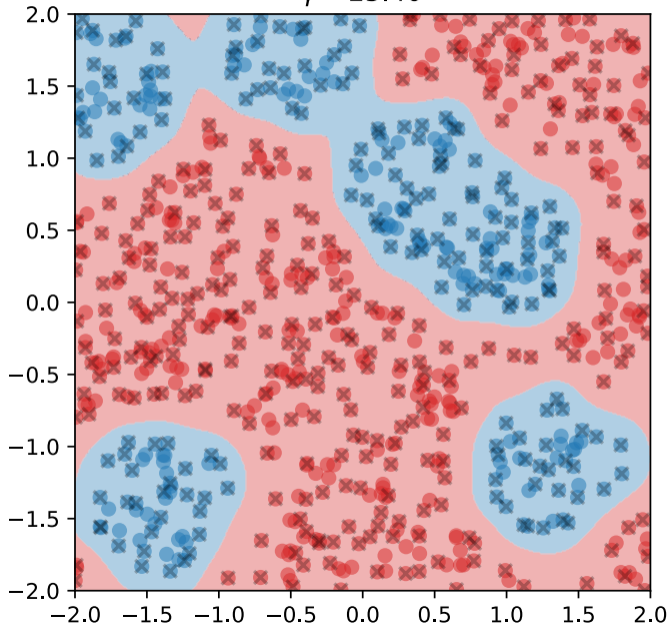
$\gamma = 2.16$



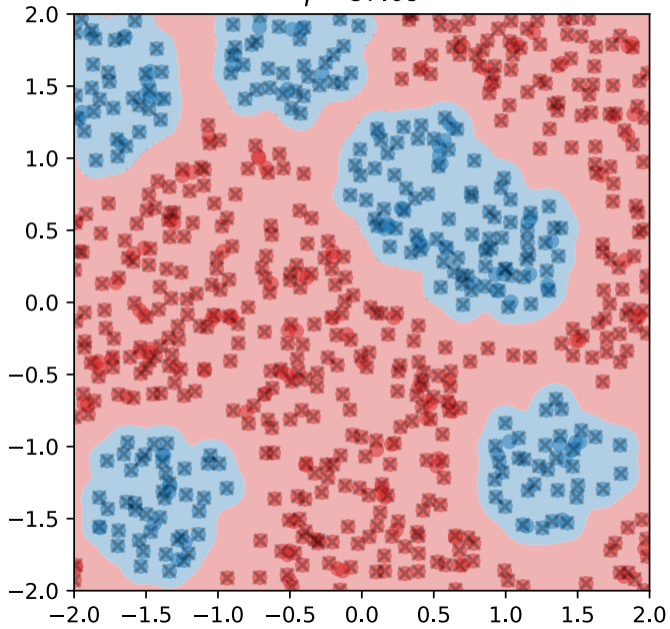
$\gamma = 7.41$



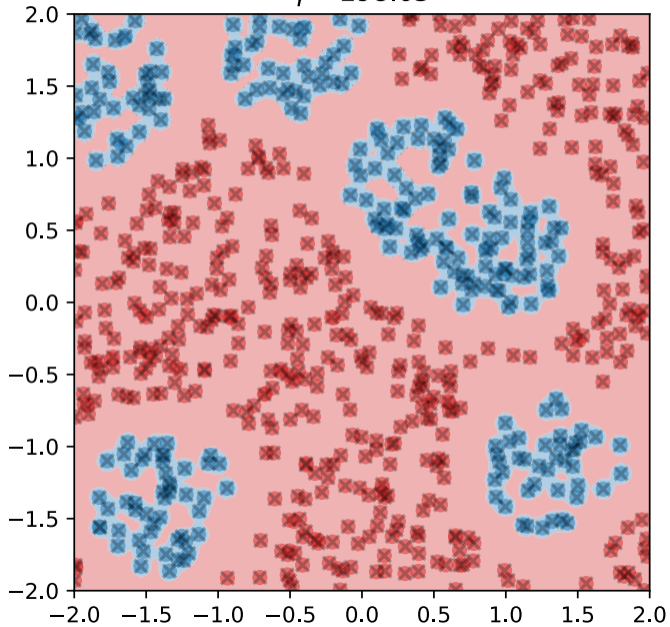
$\gamma = 25.40$



$\gamma = 87.09$



$\gamma = 298.63$



$\gamma = 1024.00$

